Question 1 (11 marks)

For each of the following type declarations:

- Give an example of **declaring and initialising a single variable** of that type.
- State what types of values that variable can hold.
- If applicable, state how the various components of the variable are arranged in memory.

(a) `typedef enum { alpha, beta = 403, gamma } Delta;` **[1 mark]**

(b) **[3 marks]**
```
struct Epsilon {
    float * const *zeta;
    char *eta[10][12];
};
```

(c) **[2 marks]**
```
typedef union {
    void (*theta)();
    void *(*iota)();
} Kappa;
```

(d) **[2 marks]**
```
typedef struct Lambda {
    struct Lambda *mu[3];
} Nu;
```

(e) **[3 marks]**
```
struct Xi {
    union {
        double pi;
        int rho;
    } sigma;
    enum { tau, upsilon } phi;
};
```

Question 2 (9 marks)

Write a C function called `loadedDice`, which takes in two integer pointer parameters and returns nothing.

The function should alternate between two behaviours:

- On every second call, it should export two sixes (via the pointer parameters).
- On every other call, it should generate two random numbers between one and six (inclusive) and export those instead.

On each call, the function should do only one of these things. On the next call, it should do the other.

Question 3 appears on the next page

Question 3 (20 marks)

Consider the following code:

```
double **a, **b;
double **c[3];
double d[] = {2.0, 3.0, 5.0, 7.0, 11.0};

a = (double**)malloc(3 * sizeof(double*));
c[2] = (double**)malloc(2 * sizeof(double*));

*c = a;
*(c + 1) = a + 2;
b = *(c + 2);

a[0] = d;
*(a + (int)*d) = d + 2;
c[0][1] = &d[2];
c[2][0] = a[1];
c[(int)**a]++;
c[2][0] = c[1][0] + 1;

a[1][0] *= b[1][0];
a[0][1] *= b[0][1];
```

Based on this:

(a) Draw a diagram showing all the pointer relationships created. **[15 marks]**

(b) Show the contents of `d` at the end. **[5 marks]**

Question 4 (20 marks)

The following function (on the next page) implements part of a face matching algorithm. The function imports an array of `Face` pointers, along with the array length. The array elements represent the same person's face from different angles.

The function scans a database of existing faces and attempts to match each one against the imported faces. If a match is found, the corresponding name is returned. Otherwise, the function returns `NULL`.

However, the program has defects! The defects are **not in the code shown here**, but rather in the functions *called* by `recognise()`.

Question 4 continues on the next page

```
1  char *recognise(Face *face[], int testFaces)
2  {
3      int i = 0, j = 0, match = FALSE, dbFaces;
4      double likelihood, threshold;
5      char *name = NULL;
6
7      Face *db = loadFaceDatabase(&dbFaces);
8      threshold = loadThreshold() * (double)testFaces;
9
10     printf("Scanning database...");
11     do
12     {
13         likelihood = 0.0;
14         for(j = 0; j < testFaces; j++)
15         {
16             likelihood += compareFaces(face[j], db[i]);
17         }
18         if(likelihood >= threshold)
19         {
20             name = lookupName(db[i]);
21         }
22         i++;
23     }
24     while(name != NULL && i < dbFaces);
25
26     free(db);
27     return name;
28 }
```

For each situation below:

- Give **two plausible hypotheses** for what might be wrong (in the code *not* shown). Briefly explain why both hypotheses fit the observations.
- Explain how you would use debugger features to rule one of them out. In particular, where would you set **breakpoints**, and which **variables** would you monitor?

State any relevant (and realistic) assumptions you make about the code not shown.

(a) The function always returning `NULL`, even when a face should match one in the database. **[5 marks]**

(b) The function displays "Scanning database…", then segfaults. **[5 marks]**

(c) The function segfaults without displaying anything. **[5 marks]**

(d) Previous debugging indicates that the `threshold` variable is behaving as expected. However, the function always returns the first name in the database. **[5 marks]**

Question 5 appears on the next page

Question 5 (40 marks)

For this question, ensure all your code conforms to the characteristics emphasised in the lectures and tutorials.

You may refer to the following standard C function prototypes:

```
FILE *fopen(const char *path, const char *mode);
int fclose(FILE *fp);
int fscanf(FILE *stream, const char *format, ...);
char *fgets(char *s, int size, FILE *stream);
int feof(FILE *stream);
int strcmp(const char *s1, const char *s2);
char *strcpy(char *dest, const char *src);
```

(a) Declare suitable C datatypes to represent each of the following sets of information (as you would do in a header file):

(i) A 2D real-numbered point (or coordinate) called `Point`. **[2 marks]**

(ii) A drawing command called `Command`. This can be one of "line" or "circle". Both commands require a colour code — a character — and at least one 2D point. Lines also require a second 2D point. Circles require a real-numbered radius.

Devise a single datatype to hold all necessary information for both these commands. (You do not need to be memory efficient, but you can if you wish.)

Your datatype must also record what the command actually is. **[3 marks]**

(iii) A sequence of drawing commands, called `Drawing`. The sequence can be any length, but will not need to change once it is created. **[3 marks]**

Question 5 continues on the next page

(b) Write a C function called `loadDrawing` to read a list of drawing commands embedded in a text file.

The first line of the file indicates how many commands follow. On subsequent lines, the line and circle commands (mentioned above) may occur in any order. They each have the following syntax:

- `LINE (`x_1`, `y_1`)-(`x_2`, `y_2`) `c
- `CIRCLE (`x`, `y`):`r c

Here, x, y and r are real numbers, while c is a non-space character representing a colour. Command names are always uppercase.

The following is an example file, representing two lines and three circles (in various different colours):

```
6
LINE (0.0, 0.0)-(3.0, 0.0) r
CIRCLE (-1.0, -1.0):0.1 g
LINE (0.0, 5.0)-(3.0, 5.0) y
CIRCLE (1.5, 2.5):1.0 b
CIRCLE (1.5, 2.5):0.5 b
```

Your function should:

- Import a filename.
- Read the file, according to the above specifications.
- Dynamically allocate the structures you designed in part (a).
- Store the command sequence in these structures.
- Return a pointer to a `Drawing`, or `NULL` if an error occurs.

If the file can be opened, you may assume that it definitely conforms to the specification.

[17 marks]

Question 5 continues on the next page

(c) Write a C function called `execDrawing`, which:

- Imports a pointer to a `Drawing` (the same information your `loadDrawing()` function returned).
- Returns nothing.
- Cycles through the sequence of line and circle commands, calling the following pre-defined functions to execute each command, thus displaying an image:

```
void colourToRGB(char code,
                 int *red, int *green, int *blue);

void drawLine(double x1, double y1,
              double x2, double y2,
              int red, int green, int blue);

void drawCircle(double x, double y,
                int red, int green, int blue);
```

The `x` and `y` parameters correspond to horizontal and vertical coordinates. You must use `colourToRGB()` to convert colour codes to more precise red-green-blue values. **[10 marks]**

(d) Write a `main` function that accepts any number of filenames as command-line parameters.

For each filename in order, `main` should:

- Use the function `readDrawing()` from part (b) to read the input file.
- Use the function `execDrawing()` from part (c) to execute the drawing and display it.
- Perform all necessary cleaning up.
- Output any relevant error messages. **[5 marks]**

— End of Examination Paper —